APPENDIX III

```
#
#                       Confidential Information
#             Limited Distribution to Authorized Persons Only
#            Created 2000 and Protected as an Unpublished Work
#                   Under the U.S.Copyright act of 1976.
#                  Copyright © 2000-2001 ARC CORES LTD
#                          All Rights Reserved
#
# SCCS release : %M% %I% %G%
#
# Description  : Script to analyse an ARC assembler file and
#                print frequency of usage stats for various
#                proposed ARC instruction formats
#
#
#
#--
================================================================================
===================--#

BEGIN {
 out = "c"
 #reg = "%r(0|1|2|3|13|14|15|16),"
 reg = "%r(0|1|2|3|13|14|15|16)([^0-9]|$)"
 regh = "%(r[0-9]+|sp|fp|gp|blink)([^0-9]|$)"
 reg01 = "%r(0|1)([^0-9]|$)"
 reg23 = "%r(2|3)([^0-9]|$)"
 reg1316 = "%r(13|14|15|16)([^0-9]|$)"
 pete = 0
 printf "" >out
}

function nxt() {
 print $0 >>out
 next
}
function nxtc() {
 print "c" $0 >>out
 next
}

$1 == "bl" {
 bl++
 if ($2 ~ /__prolog_.*/) {
  push++
  nxt()
 } else {
  calls[$2]++
  nxtc()
 }
}
$1 == "b" {
 b++
 if ($2 ~ /__epilog_.*/) {
  pop++
  nxt()
 } else {
  nxtc()
 }
}
$1 == "beq" || $1 == "bne" {
 if ($2 !~ /__epilog_.*/) {
  beq++
   nxtc()
  } else {
   nxt()
  }
 }
 $1 == "bgt" || $1 == "ble" || $1 == "bge" || $1 == "blt" {
  if ($2 !~ /__epilog_.*/) {
   bgt++
   nxtc()
  } else {
   nxt()
  }
 }
 $1 == "bhi" || $1 == "bls" || $1 == "bhs" || $1 == "blo" {
  if ($2 !~ /__epilog_.*/) {
   bhi++
   nxt()
  } else {
   nxt()
  }
 }
 $1 == "bpl" || $1 == "bmi" {
  if ($2 !~ /__epilog_.*/) {
   bpl++
   nxt()
  } else {
   nxt()
  }
 }
 $1 == "jeq" || $1 == "jne" {
  if ($2 ~ "blink") {
   beq++
   nxtc()
  }
  nxt()
 }
 $1 == "jgt" || $1 == "jle" || $1 == "jge" || $1 == "jlt" {
  if ($2 ~ "blink") {
   bgt++
   nxtc()
  }
  nxt()
 }
 $1 == "j" {
  if ($2 ~ "blink") {
    jblink++
    nxtc()
  }
  if ($2 ~ reg) {
    jr++
    nxtc()
  }
  nxt()
 }
 $1 == "jl" {
  if ($2 ~ reg) {
    jlr++
    nxtc()
  }
  nxt()
 }
 $1 == "ld" {
  if ($2 ~ reg) {
```

```
  ld++
  if ($3 == "[%fp,") {
#   ldfpa[$4]++
   ldfp++
   if (($4+0) >= -32 && ($4+0) <= -4) {
    ldfp32++
    nxtc()
   }
   nxt()
  }
  if ($3 == "[%sp,") {
#   ldspa[$4]++
   ldsp++
   nxt()
  }
  if ($3 == "[%gp,") {
   ldgp++
   nxtc()
  }
  if ($3 ~ reg) {
#   ldra[$4]++
   ldr++
   if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 64)) {
    ldr64++
    nxtc()
   }
   if (pete) {
    if ($3 ~ /\]/ || ($3 ~ reg01 && ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) <
128))) {
     ldr64p++
     nxtc()
    }
     if ($3 ~ /\]/ || ($3 ~ reg23 && ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) <
64))) {
     ldr64p++
     nxtc()
    }
     if ($3 ~ /\]/ || ($3 ~ reg1316 && ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) <
32))) {
     ldr64p++
     nxtc()
    }
   }
   if ($4 ~ reg) {
    ldabc++
    nxtc()
   }
   nxt()
  }
 }
 nxt()
}
$1 == "ldw" {
 if ($2 ~ reg) {
  ldw++
  if ($3 == "[%fp,") {
   ldwfp++
   if (($4+0) >= -32 && ($4+0) <= -4) {
    ldwfp32++
    nxtc()
   }
   nxt()
  }
```

```
      if ($3 == "[%sp,") {
       ldwsp++
       nxt()
      }
      if ($3 == "[%gp,") {
       ldwgp++
       nxtc()
      }
      if ($3 ~ reg) {
       ldwr++
       if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 32)) {
        ldwr32++
        nxtc()
       }
       if ($4 ~ reg) {
        ldwabc++
        nxt()
       }
       nxt()
      }
     }
    nxt()
   }
   $1 == "ldb" {
    if ($2 ~ reg) {
     ldb++
     if ($3 == "[%fp,") {
      ldbfp++
      if (($4+0) >= -32 && ($4+0) <= -4) {
       ldbfp32++
       nxt()
      }
      nxt()
     }
     if ($3 == "[%sp,") {
      ldbsp++
      nxt()
     }
     if ($3 == "[%gp,") {
      ldbgp++
      nxt()
     }
     if ($3 ~ reg) {
      ldbr++
      if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 16)) {
       ldbr16++
       nxtc()
      }
      if ($4 ~ reg) {
       ldbabc++
       nxt()
      }
      nxt()
     }
    }
    nxt()
   }
   /st.%blink, \[%sp, 4\]/ {
    stblink++
    nxtc()
    }
   $1 == "st" {
    if ($2 ~ reg) {
```

```
  st++
  if ($3 == "[%fp,") {
#   stfpa[$4]++
   stfp++
   if (($4+0) >= -32 && ($4+0) <= -4) {
    stfp32++
    nxtc()
   }
   nxt()
  }
  if ($3 == "[%sp,") {
#   stspa[$4]++
   stsp++
   nxt()
  }
  if ($3 == "[%gp,") {
   stgp++
   nxt()
  }
  if ($3 ~ reg) {
#   stra[$4]++
   str++
   if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 64)) {
    str64++
    nxtc()
   }
   nxt()
  }
 }
 nxt()
}
$1 == "stw" {
 if ($2 ~ reg) {
  stw++
  if ($3 == "[%fp,") {
#   stwfpa[$4]++
   stwfp++
   if (($4+0) >= -32 && ($4+0) <= -4) {
     stwfp32++
   nxt()
   }
   nxt()
  }
  if ($3 == "[%sp,") {
#   stwspa[$4]++
   stwsp++
   nxt()
  }
  if ($3 == "[%gp,") {
   stwgp++
   nxt()
  }
  if ($3 ~ reg) {
#   stwra[$4]++
   stwr++
   if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 16)) {
    stwr16++
    nxtc()
   }
   nxt()
  }
 }
 nxt()
```

```
}
$1 == "stb" {
 if ($2 ~ reg) {
  stb++
  if ($3 == "[%fp,") {
#   stbfpa[$4]++
   stbfp++
   if (($4+0) >= -32 && ($4+0) <= -4) {
     stbfp32++
   nxt()
   }
   nxt()
  }
  if ($3 == "[%sp,") {
#   stbspa[$4]++
   stbsp++
   nxt()
  }
  if ($3 == "[%gp,") {
   stbgp++
   nxt()
  }
  if ($3 ~ reg) {
#   stbra[$4]++
   stbr++
   if ($3 ~ /\]/ || ($4 ~ /^[0-9]/ && ($4+0) >= 0 && ($4+0) < 8)) {
    stbr8++
    nxtc()
   }
   nxt()
  }
 }
 nxt()
}
$1 == "mov.f" {
 if ($2 == "0," && $3 ~ reg) {
  movf0r++
  nxtc()
 } if ($2 == "0," && $3 ~ regh) {
  movf0h++
  nxtc()
 }
 nxt()
}
$1 == "mov" {
 if ($3 ~ /^-?[0-9]/) {
  movi++
  movia[$3]++
  if ($2 ~ reg) {
   if ($3 >= 0 && $3 < 64) {
    movi64++
    nxtc()
   }
   if (pete) {
    if ($2 ~ reg01 && $3 >= 0 && $3 < 128) {
     movi64p++
     nxtc()
    }
    if ($2 ~ reg23 && $3 >= 0 && $3 < 64) {
     movi64p++
     nxtc()
    }
    if ($2 ~ reg1316 && $3 >= 0 && $3 < 32) {
      movi64p++
      nxtc()
     }
    }
    if ($3 < -256 || $3 > 255) {
     ldrpc++
     nxtc()
    }
   }
   nxt()
  }
  if ($3 ~ reg) {
   if ($2 ~ reg) {
    movr++
    nxtc()
   }
  }
  if ($2 ~ reg) {
   if ($3 ~ regh) {
    movrh++
    nxtc()
   }
  }
  if ($2 ~ regh) {
   if ($3 ~ reg) {
    movhr++
    nxtc()
   }
  }
  if ($3 !~ /^%/ && $2 ~ reg) {
   ldrpc++
   nxtc()
  }
  nxt()
 }
 $1 == "add" {
  if ($2 == $3 || $2 == ($3 ",") || $2 == ($4 ",")) {
   if ($4 ~ /^-?[0-9]/) {
    addi++
#   addia[$4]++
    if ($3 ~ reg) {
     if ($4 >= -32 && $4 < 0) {
      subi32++
      nxtc()
     }
     if ($4 >= 0 && $4 < 32) {
      addi32++
      nxtc()
     }
    }
   }
   if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
    addaab++
    nxtc()
   }
   if ($2 ~ reg && $3 ~ reg && $4 ~ regh) {
    addrrh++
    nxtc()
   }
   if ($2 ~ reg && $3 ~ regh && $4 ~ reg) {
    addrrh++
    nxtc()
   }
```

```
    }
    if ($4 ~ /^-?[0-9]/) {
     if ($2 ~ reg) {
      if ($3 ~ reg) {
       if ($4 >= -8 && $4 < 0) {
        subabi8++
        nxtc()
       }
       if ($4 >= 1 && $4 <= 8) {
        addabi8++
        nxtc()
       }
      }
      if ($3 ~ "%fp") {
       if ($4 >= -32 && $4 < 0) {
        addfpi32++
        nxtc()
       }
      }
      if ($3 ~ /^%r([12][0-9])/ && $4 >= -512 && $4 < 512) {
       addrpc++
       nxtc()
      }
     }
     nxt()
    }
    if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
     addrrr++
     nxtc()
    }
   }
   $1 == "sub" {
    if ($4 ~ /^-?[0-9]/) {
     subi++
     if ($2 == $3) {
   #   subia[$4]++
      if ($3 ~ reg) {
       if ($4 >= -32 && $4 < 0) {
        addi32++
        nxtc()
       }
       if ($4 >= 0 && $4 < 32) {
        subi32++
        nxtc()
       }
      }
     }
     if ($2 ~ reg) {
      if ($3 ~ reg) {
       if ($4 >= -8 && $4 < 0) {
        addabi8++
        nxtc()
       }
       if ($4 >= 1 && $4 < 8) {
        subabi8++
        nxtc()
       }
      }
     }
     nxt()
    }
    if ($2 == $3 && $2 == ($4 ",")) {
     if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
```

```
     subaaa++
     nxtc()
    }
    if ($2 ~ regh && $3 ~ regh && $4 ~ regh) {
     subhhh++
     nxtc()
    }
   }
   if ($2 ~ reg) {
    subr++
    if ($2 == $3) {
     if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
      subaab++
      nxtc()
     }
     if ($2 ~ reg && $3 ~ reg && $4 ~ regh) {
      subrrh++
      nxtc()
     }
     if ($2 ~ reg && $3 ~ regh && $4 ~ reg) {
      subrrh++
      nxtc()
     }
    }
    if ($3 ~ reg && $4 ~ reg) {
     subrrr++
     nxtc()
    }
    nxt()
   }
  }
  $1 == "sub.f" {
   if ($2 == "0,") {
    if ($3 ~ reg && $4 ~ reg) {
     cmpr++
     nxtc()
    }
    if ($4 ~ /^-?[0-9]/) {
     cmpi++
  #  cmpia[$4]++
     if ($3 ~ reg) {
      if ($4 >= 0 && $4 < 64) {
       cmpi64++
       nxtc()
      }
      if (pete) {
       if ($3 ~ reg01 && $4 >= 0 && $4 < 128) {
        cmpi64p++
        nxtc()
       }
       if ($3 ~ reg23 && $4 >= 0 && $4 < 64) {
        cmpi64p++
        nxtc()
       }
       if ($3 ~ reg1316 && $4 >= 0 && $4 < 32) {
        cmpi64p++
        nxtc()
       }
      }
     }
    nxt()
   }
   if ($3 ~ reg) {
    if ($4 ~ regh) {
     cmprh++
     nxtc()
    }
   }
   if ($3 ~ regh) {
    if ($4 ~ reg) {
     cmphr++
     nxtc()
    }
   }
  }
  nxt()
 }
 $1 == "sub.ne" {
  if ($2 == $3 && $2 == ($4 ",")) {
   if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
    subneaaa++
    nxtc()
   }
  }
  nxt()
 }
 $1 == "sub.eq" {
  if ($2 == $3 && $2 == ($4 ",")) {
   if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
    subeqaaa++
    nxtc()
   }
  }
  nxt()
 }
 $1 == "asl" {
  if ($4 ~ /^-?[0-9]/) {
   asli++
   if ($2 == $3) {
 #   aslia[$4]++
    if ($3 ~ reg) {
     if ($4 >= 1 && $4 <= 8) {
      asli8++
     }
     if ($4 >= 1 && $4 < 32) {
      asli32++
     }
     nxtc()
    }
   }
   if ($2 ~ reg) {
    if ($3 ~ reg && $4 >= 2 && $4 < 3) {
     aslab2++
     nxtc()
    }
   }
   nxt()
  }
  if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
   aslaab++
   nxtc()
  }
  if ($2 ~ reg && $3 ~ reg && $4 !~ reg) {
   aslab1++
   nxtc()
  }
```

```
}
$1 == "asr" {
 if ($4 ~ /^-?[0-9]/) {
  asri++
  if ($2 == $3) {
#   asria[$4]++
   if ($3 ~ reg) {
    if ($4 >= 1 && $4 <= 8) {
     asri8++
    }
    if ($4 >= 1 && $4 < 32) {
     asri32++
    }
    nxtc()
   }
  }
  if ($2 ~ reg) {
   if ($3 ~ reg && $4 >= 2 && $4 < 3) {
    asrab2++
    nxtc()
   }
  }
  nxt()
 }
 if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
  asraab++
  nxtc()
 }
 if ($2 ~ reg && $3 ~ reg && $4 !~ reg) {
  asrab1++
  nxtc()
 }
}
$1 == "lsr" {
 if ($4 ~ /^-?[0-9]/) {
  lsri++
  if ($2 == $3) {
#   lsria[$4]++
   if ($3 ~ reg) {
    if ($4 >= 1 && $4 <= 8) {
     lsri8++
    }
    if ($4 >= 1 && $4 < 32) {
     lsri32++
    }
    nxtc()
   }
  }
  if ($2 ~ reg) {
   if ($3 ~ reg && $4 >= 2 && $4 < 3) {
    lsrab2++
    nxtc()
   }
  }
  nxt()
 }
 if ($4 ~ reg && $2 ~ reg && $3 ~ reg) {
  lsraab++
  nxtc()
 }
 if ($2 ~ reg && $3 ~ reg && $4 !~ reg) {
  lsrab1++
  nxtc()
```

```
    }
   }
   $1 == "mul64" {
    if ($2 == "0,") {
     if ($4 ~ /^-?[0-9]/) {
      muli++
#      mulia[$4]++
       if ($3 ~ reg) {
        if ($4 >= 0 && $4 < 32) {
         muli32++
         nxtc()
        }
       }
     }
     if ($3 ~ reg && $4 ~ reg) {
      mul0ab++
      nxtc()
     }
    }
    nxt()
   }
   $1 == "and.f" {
    if ($2 == "0,") {
     if ($4 ~ /^-?[0-9]/) {
      andfi++
#      andfia[$4]++
       if ($3 ~ reg) {
        if ($4 >= 0 && $4 < 32) {
         andfi32++
         nxtc()
        }
       }
     }
     if ($3 ~ reg && $4 ~ reg) {
      andfab++
      nxtc()
     }
    }
    nxt()
   }
   $1 == "and" {
    if ($2 == $3 || $2 == ($3 ",") || $2 == ($4 ",")) {
     if ($4 ~ /^-?[0-9]/) {
      andi++
#      andia[$4]++
       if ($3 ~ reg) {
        if ($4 >= 0 && $4 < 32) {
         andi32++
         nxtc()
        }
       }
     }
     if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
      andaab++
      nxtc()
     }
    }
    if ($2 ~ reg && $3 ~ reg && $4 ~ reg) {
     andrrr++
     nxt()
    }
   }
   $1 == "extb" {
```

```
 if ($2 == ($3 ",")) {
  if ($2 ~ reg && $3 ~ reg) {
   extbr++
   nxtc()
  }
 }
 nxt()
}
$1 == "extw" {
 if ($2 == ($3 ",")) {
  if ($2 ~ reg && $3 ~ reg) {
   extwr++
   nxtc()
  }
 }
 nxt()
}
$1 == "sexb" {
 if ($2 == ($3 ",")) {
  if ($2 ~ reg && $3 ~ reg) {
   sexbr++
   nxtc()
  }
 }
 nxt()
}
$1 == "sexw" {
 if ($2 == ($3 ",")) {
  if ($2 ~ reg && $3 ~ reg) {
   sexwr++
   nxtc()
  }
 }
 nxt()
}
($2 == $3 || $2 == ($3 ",") || $2 == ($4 ",")) {
 if ($1 == "add" || $1 == "sub" || $1 == "and" || $1 == "or" || $1 == "xor" ||
$1 == "asl" || $1 == "asr" || $1 == "lsr") {
  if ($2 ~ reg) {
   if ($2 == $3) {
    if ($4 ~ reg) {
     opaab[$1]++
     nxtc()
    }
   } else {
    if ($3 ~ reg && $2 == ($4 ",")) {
     opaab[$1]++
     nxtc()
    }
   }
  }
 }
}

{
 nxt()
# print $0
}

END {
if (1) {
 OFS = "\t"
#  print "\nopaab"
```

```
  for (i in opaab) {
   if (i == "add" || i == "sub" || i == "and" || i == "or" || i == "xor" || i ==
"asl" || i == "asr" || i == "lsr") {
    print i, opaab[i], int(opaab[i]*1000/NR)/10
   }
  }
# print "\nldfpa"
# for (i in ldfpa) print i, ldfpa[i]
# print "\nstfpa"
# for (i in stfpa) print i, stfpa[i]
# print "\nldr0a"
# for (i in ldr0a) print i, ldr0a[i]
# print "\nmovia"
# for (i in movia) print i, movia[i]
# print "\naddia"
# for (i in addia) print i, addia[i]
# print "\nsubia"
# for (i in subia) print i, subia[i]
# print "\ncmpia"
# for (i in cmpia) print i, cmpia[i]

 for (i in calls) {
#  print i, calls[i]
  if (calls[i] > 1) {
   calls2 += (calls[i]-2)
  }
  callsall += calls[i]
 }
# print "callsall", callsall, int(callsall*1000/NR)/10
# print "calls2", calls2, int(calls2*1000/NR)/10

# bl = calls2
 bl = bl - push
 b = b - pop
 print "bl", bl, int(bl*1000/NR)/10
# print "push", push, int(push*1000/NR)/10

 print "b", b, int(b*1000/NR)/10
# print "pop", pop, int(pop*1000/NR)/10
 print "beq", beq, int(beq*1000/NR)/10
 print "bgt", bgt, int(bgt*1000/NR)/10
 print "bhi", bhi, int(bhi*1000/NR)/10
 print "bpl", bpl, int(bpl*1000/NR)/10

 print "stblink", stblink, int(stblink*1000/NR)/10
 print "jblink", jblink, int(jblink*1000/NR)/10
 print "jr", jr, int(jr*1000/NR)/10
 print "jlr", jlr, int(jlr*1000/NR)/10

 print "movr", movr, int(movr*1000/NR)/10
 print "movf0r", movf0r, int(movf0r*1000/NR)/10
 print "movf0h", movf0h, int(movf0h*1000/NR)/10
 print "movrh", movrh, int(movrh*1000/NR)/10
 print "movhr", movhr, int(movhr*1000/NR)/10

 print "cmprh", cmprh, int(cmprh*1000/NR)/10
 print "cmphr", cmphr, int(cmphr*1000/NR)/10
 print "cmpr", cmpr, int(cmpr*1000/NR)/10

 print "cmpi64", cmpi64, int(cmpi64*1000/NR)/10
 print "cmpi64p", cmpi64p, int(cmpi64p*1000/NR)/10
 print "movi64", movi64, int(movi64*1000/NR)/10
 print "movi64p", movi64p, int(movi64p*1000/NR)/10
 print "addi32", addi32, int(addi32*1000/NR)/10
 print "subi32", subi32, int(subi32*1000/NR)/10

 print "addabi8", addabi8, int(addabi8*1000/NR)/10
 print "subabi8", subabi8, int(subabi8*1000/NR)/10

 print "subneaaa", subneaaa, int(subneaaa*1000/NR)/10
 print "subeqaaa", subeqaaa, int(subeqaaa*1000/NR)/10

 print "subhhh", subhhh, int(subhhh*1000/NR)/10
 print "subaaa", subaaa, int(subaaa*1000/NR)/10
 print "subaab", subaab, int(subaab*1000/NR)/10
 print "subrrr", subrrr, int(subrrr*1000/NR)/10
 print "addaab", addaab, int(addaab *1000/NR)/10
 print "addrrr", addrrr, int(addrrr *1000/NR)/10
 print "addrrh", addrrh, int(addrrh *1000/NR)/10

 print "asli8", asli8, int(asli8*1000/NR)/10
# print "asli32", asli32, int(asli32*1000/NR)/10
 print "aslab1", aslab1, int(aslab1*1000/NR)/10
 print "aslab2", aslab2, int(aslab2*1000/NR)/10
 print "aslaab", aslaab, int(aslaab*1000/NR)/10

 print "asri8", asri8, int(asri8*1000/NR)/10
# print "asri32", asri32, int(asri32*1000/NR)/10
 print "asrab1", asrab1, int(asrab1*1000/NR)/10
 print "asrab2", asrab2, int(asrab2*1000/NR)/10
 print "asraab", asraab, int(asraab*1000/NR)/10

 print "lsri8", lsri8, int(lsri8*1000/NR)/10
# print "lsri32", lsri32, int(lsri32*1000/NR)/10
 print "lsrab1", lsrab1, int(lsrab1*1000/NR)/10
 print "lsrab2", lsrab2, int(lsrab2*1000/NR)/10
 print "lsraab", lsraab, int(lsraab*1000/NR)/10

 print "andi32", andi32, int(andi32*1000/NR)/10
 print "andfi32", andfi32, int(andfi32*1000/NR)/10
 print "andaab", andaab, int(andaab *1000/NR)/10
 print "andfab", andfab, int(andfab *1000/NR)/10

 print "mul0ab", mul0ab, int(mul0ab *1000/NR)/10
 print "muli32", muli32, int(muli32 *1000/NR)/10

 print "ldabc", ldabc, int(ldabc *1000/NR)/10
 print "ldbabc", ldbabc, int(ldbabc *1000/NR)/10
 print "ldwabc", ldwabc, int(ldwabc *1000/NR)/10
 print "ldr64", ldr64, int(ldr64 *1000/NR)/10
 print "ldr64p", ldr64p, int(ldr64p *1000/NR)/10
 print "ldwr32", ldwr32, int(ldwr32 *1000/NR)/10
 print "ldbr16", ldbr16, int(ldbr16 *1000/NR)/10
 print "str64", str64, int(str64 *1000/NR)/10
 print "stbr8", stbr8, int(stbr8 *1000/NR)/10
 print "stwr16", stwr16, int(stwr16 *1000/NR)/10

 print "ldrpc", ldrpc, int(ldrpc *1000/NR)/10
 print "addrpc", addrpc, int(addrpc *1000/NR)/10

 print "ldfp32", ldfp32, int(ldfp32*1000/NR)/10
 print "stfp32", stfp32, int(stfp32*1000/NR)/10
 print "addfpi32", addfpi32, int(addfpi32*1000/NR)/10

 print "ldgp", ldgp, int(ldgp*1000/NR)/10
 print "stgp", stgp, int(stgp*1000/NR)/10

 print "extbr", extbr, int(extbr*1000/NR)/10
 print "extwr", extwr, int(extwr*1000/NR)/10
 print "sexbr", sexbr, int(sexbr*1000/NR)/10
 print "sexwr", sexwr, int(sexwr*1000/NR)/10

# print "movi", movi, "movi64", movi64, "movi128", movi128
# print "addi", addi, "addi32", addi32, "addi64", addi64, "addi128", addi128
# print "subi", subi, "subi32", subi32, "subi64", subi64, "subi128", subi128
 }
}
#function p(a, b) {
# print "a", b, int(b*100/NR)
#}

#/(j|jl|b|bl)(ge|gt|le|lt|ne|eq|pl|mi|hi|hs|lo|ls)?\.d/ {
#  stored = $0
#  sub(/\.d/, "", stored)
#  getline
#  print $0
#  print stored
#  nxtc()
#}
#
#{ print $0 }
```